Asignatura: Fundamentos de algoritmia
Evaluación continua algoritmos recursivos. Cuestionario.

Profesor: Isabel Pita.

Nombre del alumno:

1. (2 puntos) Dado el problema de la búsqueda binaria, en el que nos piden indicar si un valor pertenece a un vector:

   a) Escribe la precondición del problema, es decir las condiciones necesarias sobre los datos de entrada para que se ejecute correctamente el algoritmo.

   b) Escribe una definición recursiva del problema. Observa que se pide una *definición recursiva*, no la implementación en un lenguaje de programación.

   c) Escribe la llamada inicial a la función para que se busque el valor en todo el vector.

   d) Indica la recurrencia que permite calcular el coste en el caso peor del algoritmo de búsqueda binaria e indica el coste final del algoritmo.

   Respuesta:

   a) La precondición de la función `bool BuscaBin(v,x)`.
   P: { $v.size >= 0 \wedge \forall k : 0 \leq k < v.size - 1 : v[k] <= v[k+1]$ }. El vector tiene que estar ordenado.
   La precondición de la función `bool BuscaBin(v,x,ini, fin)`.
   P: { $0 <= ini <= fin <= v.size() \wedge \forall k : ini \leq k < fin - 1 : v[k] <= v[k+1]$ }.
   Observad que el vector puede ser vacío. El resultado de la búsqueda de un elemento en un vector vacío es siempre false.

   b)

$$busqBin(v, x, ini, fin) = \begin{cases} false & if \quad ini == fin \textit{(caso base, vector vacío)} \\ true & if \quad ini < fin \wedge v[(ini + fin - 1)/2] == x \textit{(caso base)} \\ busqBin(v, x, ini, m) & if \quad ini < fin \wedge v[(ini + fin - 1)/2] > x \\ busqBin(v, x, m + 1, fin) & if \quad ini < fin \wedge v[(ini + fin - 1)/2] < x \end{cases}$$

   c) busqBin(v,x,0,v.size()); donde v es el vector de entrada y x el valor que estamos buscando.

   d)

$$T(n) = \begin{cases} c_0 & if \quad ini >= fin \\ T(n/2) + c_1 & if \quad ini < fin \end{cases}$$

   El coste final del algoritmo es $\mathcal{O}(\log n)$, siendo n el número de elementos del vector de entrada ($n = fin - ini$).
   El desplegado se puede consultar en el cuaderno de problemas de complejidad de algoritmos recursivos en el campus.

2. (1 punto) Indica cuál es la mejor complejidad que puede tener un algoritmo de ordenación basado en comparaciones, si los valores del vector son números enteros cualesquiera. Indica si conoces algún algoritmo con esta complejidad y en caso afirmativo dí cuál o cuales son.

   Respuesta:

   La mejor complejidad que se puede obtener con un algoritmo de ordenación basado en comparaciones es $\mathcal{O}(n \log n)$, siendo n el tamaño del vector.

   Se han estudiado los algoritmos de quicksort y mergesort con esta complejidad. Quicksort tiene complejidad $\mathcal{O}(n \log n)$ en el caso equilibrado y complejidad $\mathcal{O}(n^2)$ en el caso peor.

   Mergesort tiene complejidad $\mathcal{O}(n \log n)$ en el caso peor y en el caso medio.

3. (1 punto) Explica porque la constante multiplicativa del algoritmo del mergesort es alta.

Respuesta:

Se debe principalmente a que el algoritmo de la mezcla ordenada, utilizado para mezclar las dos partes ordenadas del vector utiliza un vector auxiliar para construir la mezcla. Los elementos deben copiarse en el vector auxiliar y volverse a copiar en el vector original una vez mezclados.

4. (2 punto) Indica el mejor orden de complejidad con que se pueden resolver los siguientes problemas:

Dada una secuencia de números consecutivos, en la que falta un valor, encontrar el valor que falta.	
Mergesort	
Par de puntos mas cercanos	
Encontrar el mínimo de una serie de valores que representan una curva cóncava	
Contar las inversiones de una secuencia de valores	
Dada una secuencia de valores comprobar si algún valor coincide con su posición en la secuencia	

Respuesta:

Dada una secuencia de números consecutivos, en la que falta un valor, encontrar el valor que falta.	$\mathcal{O}(\log n)$
Mergesort	$\mathcal{O}(n \log n)$
Par de puntos mas cercanos	$\mathcal{O}(n \log n)$
Encontrar el mínimo de una serie de valores que representan una curva cóncava	$\mathcal{O}(\log n)$
Contar las inversiones de una secuencia de valores	$\mathcal{O}(n \log n)$
Dada una secuencia de valores comprobar si algún valor coincide con su posición en la secuencia	$\mathcal{O}(\log n)$

siendo n el tamaño del vector de entrada.

Estos problemas están propuestos en el juez de clase o resueltos en el cuaderno de problemas:

a) Dada una secuencia de números consecutivos, en la que falta un valor, encontrar el valor que falta. - Problema 40 Fuga de prisión.

b) Mergesort- Resuelto en el cuaderno de problemas

c) Par de puntos mas cercanos - Resuelto en el cuaderno de problemas

d) Encontrar el mínimo de una serie de valores que representan una curva cóncava - Problema 39

e) Contar las inversiones de una secuencia de valores - Problema 46 Battle star galactica

f) Dada una secuencia de valores comprobar si algún valor coincide con su posición en la secuencia - Problema 42 el juego del bongo

5. (2 punto) Dado el algoritmo del *quicksort*

a) Indica cuál es la recurrencia que permite calcular su coste en el caso peor y en el caso equilibrado.

b) Indica para cada caso su orden de complejidad.

c) Indica sobre que vectores se obtiene el caso peor y sobre cuales se obtiene el caso equilibrado

Respuesta:

En el caso peor, el pivote se sitúa siempre en un extremo del vector, dando lugar a una llamada recursiva con el vector vacío y otra llamada con $n-1$ elementos, siendo n el número de elementos del vector que se está ordenando ($n = fin - ini$) . La recurrencia es:

$$T(n) = \begin{cases} c_0 & if \quad ini == fin \\ T(n-1) + n & if \quad ini < fin \end{cases}$$

cuyo coste está en $\mathcal{O}(n^2)$.

En el caso medio el pivote se sitúa aproximadamente en el centro del vector, dando lugar a dos llamadas recursivas cada una con aproximadamente la mitad de los elementos del vector. La recurrencia es:

$$T(n) = \begin{cases} c_0 & if \quad ini == fin \\ 2T(n/2) + n & if \quad ini < fin \end{cases}$$

cuyo coste está en $\mathcal{O}(n \log n)$.

En ambos casos el coste n de la parte recursiva corresponde al algoritmo de partición.

El caso peor se obtiene cuando el vector está ordenado en orden creciente o decreciente. El caso equilibrado se obtiene cuando los valores se distribuyen uniformemente en el vector, situándose el pivote aproximadamente en el centro del vector. Por ejemplo con los valores del 1 al 6 el caso peor sería 1,2,3,4,5,6 y el caso equilibrado 4,3,5,2,6,1

Observad que si el vector tiene todos sus elementos iguales, empleando el algoritmo de partición en tres pedazos el coste es constante, ya que el vector queda dividido en tres partes, las dos de los extremos vacías y la del centro con todos los valores iguales.

6. (2 puntos) Dado el problema de calcular la distancia mínima entre un conjunto de puntos del plano. Si el algoritmo se resuelve con la técnica de divide y vencerás

   a) Indica los casos base que se deben considerar y justifica porque son necesarios.

   b) Indica los pasos que se deben dar para resolver el problema.

   Respuesta:

   a) Se necesita caso base para dos puntos y para tres puntos. La función debe tener como precondición que el conjunto de puntos sea al menos dos, ya que no tiene sentido buscar la distancia mínima entre dos puntos si no existen dos puntos. El caso base de tres puntos es necesario porque un vector de 3 puntos se divide en una parte con dos puntos y una parte con un punto. La llamada recursiva con la parte que solo tiene un punto no tiene sentido.

   b) Los pasos son:

      1) Ordenar el vector por la componente x antes de comenzar la función recursiva.
      2) Hacer dos llamadas recursivas cada una con la mitad de los elementos del vector
      3) Calcular la distancia mínima entre las dos distancias que han devuelto las llamadas recursivas.
      4) Las llamadas recursivas deben devolver los puntos del vector ordenados por la componente y. Hacer la mezcla de las dos mitades del vector para que los puntos del vector queden ordenados por la componente y.
      5) filtrar los puntos que se encuentran a una distancia menor que la mínima calculada en el punto 3 del punto medio del vector.
      6) Comprobar si la distancia de cada punto con los cinco siguientes del vector filtrado es menor que la distancia mínima del punto 3.